ARL-TR-7369 ● AUG 2015

US Army Research Laboratory

Optimizing Performance of Scientific Visualization Software to Support Frontier-Class Computations

by Richard C Angelini

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-7369 ● AUG 2015

US Army Research Laboratory

Optimizing Performance of Scientific Visualization Software to Support Frontier-Class Computations

by Richard C Angelini
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
August 2015	Final	December 2014–May 2015

4. TITLE AND SUBTITLE
Optimizing Performance of Scientific Visualization Software to Support Frontier-Class Computations

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Richard C Angelini

5d. PROJECT NUMBER
R.0015490.1

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIH-S
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-7369

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Department of Defense (DOD) High-Performance Computing (HPC) Modernization Program has initiated a new class of funded research initiatives known as “Frontier” projects. These multiyear projects are devised to leverage using multidisciplinary teams to solve problems which are orders-of-magnitude larger than typical HPC-sized challenge projects. Current scientific visualization techniques are more than adequate for solving a vast majority of the datasets generated; however, these Frontier project datasets with anticipated mesh sizes of more than 10 billion cells offer a variety of challenges for data analysis applications. In this project, EnSight was evaluated against several Frontier-sized datasets to determine if this commercial software application could be optimized to process this data. EnSight has already been established as an essential application within the DOD for interactive data analysis of large HPC simulation results. This particular project focused on not only being able to produce results from very large Frontier-class computational data results but to also optimize overall performance to reduce the amount of time required to visualize those results.

15. SUBJECT TERMS
EnSight, high-performance computing, distributed rendering, large datasets, performance optimization

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Richard C Angelini
Unclassified	Unclassified	Unclassified	UU	46	19b. TELEPHONE NUMBER (Include area code) 410-278-6266

Standard Form 298 (Rev 8/98)
Prescribed by ANSI Std Z39 18

Contents

List of Figures iv

Acknowledgments v

1. Introduction 1

2. Problem Description 3

3. Methodology 5

4. EnSight Configuration 7

5. Results 11

6. Conclusion 13

7. References 15

Appendix A. Simple 4-node Batch Example Script 17

Appendix B. Simple 5-node Batch Example Script 21

Appendix C. Mixed-Node Batch Script Example Script 25

Appendix D. Cray Distributed Rendering Example Script 29

Bibliography 35

List of Symbols, Abbreviations, and Acronyms 37

Distribution List 38

List of Figures

Fig. 1 Isosurface of Q_Criterion function and shock wave boundary generated using EnSight. Data courtesy of Dr Nicholas Bisek, US Air Force Research Laboratory/Aerospace Systems Directorate.....4

Fig. 2 The EnSight user interface provides an opportunity to set the step value for striding through a structured dataset.....6

Fig. 3 Basic EnSight 4-node client-server configuration.....8

Fig. 4 EnSight 5-node client-server configuration.....9

Fig. 5 EnSight 5-node client-server configuration using hardware-enabled client node.....9

Fig. 6 EnSight distributed rendering configuration using standard compute nodes.....10

Fig. 7 EnSight distributed rendering configuration using hardware-enabled graphics nodes.....11

Fig. 8 Timing results from various EnSight releases.....12

Fig. 9 Time to render a 4K image using various configurations against 3 sample datasets.....12

Fig. 10 Timing results from a variety of DOD/HPCMP supercomputer resources.....13

Acknowledgments

This work was supported in part by a grant of computer time and resources by the Department of Defense (DOD) High-Performance Computing (HPC) Modernization Program. The author thanks Dr Michael Stephens (US Army Corps of Engineers Engineer Research and Development Center) for general assistance with calculation of Q_critierion and shock wave boundary isosurfaces associated with the sample datasets; Dr Nicholas Bisek (US Air Force Research Laboratory, Aerospace Systems Directorate, Hypersonic Sciences Branch) for providing sample datasets and permission to use an image of Q_Criterion isosurface for this report; Dr Anders Grimsrud (Computational Engineering, Inc.) for extensive collaboration to test/debug EnSight software; Mr David Pratt (Secure Missions Solutions, Inc.) for assistance with accessing graphics processing unit (GPU)-enabled nodes on the HPC utility server systems via the Portable Batch System (PBS) batch job queuing system; and Mr Philip Matthews (Lockheed-Martin, US Army Research Laboratory DOD Supercomputing Resource Center system administrator) for assistance with enabling PBS to access GPU-enabled resources to support hardware rendering.

INTENTIONALLY LEFT BLANK.

1. Introduction

EnSight, ParaView and VisIt are stable production-quality software packages supported by the Department of Defense (DOD) High-Performance Computing Modernization Program (HPCMP) for both interactive and batch processing of large HPC-sized datasets. In recent years, efforts have been made by the HPCMP Data Analysis and Assessment Center (DAAC) personnel to optimize using these tools by providing an easy-to-use graphical job-launching interface to allow for interactive client-server interrogation of very large datasets in an interactive session to the HPC systems. (Angelini 2011, 2012) These applications provide robust toolsets for data analysis and animation of results while leveraging the computational processing power of the HPC systems. Until recently however, batch processing of HPC data has not been given adequate attention to leverage the full capabilities of the HPC resources to provide optimal turnaround time for data analysis of very large datasets.

The DOD HPCMP has initiated a new class of funded research initiatives known as "Frontier" projects. These multiyear projects are devised to leverage using multidisciplinary teams, HPCMP supercomputer resources, and advanced modeling techniques to solve problems which are an order-of-magnitude larger than a typical HPC-sized challenge project. Current scientific visualization techniques are more than adequate for solving a vast majority of the datasets generated on the DOD/HPC resources. However, the Frontier project datasets with anticipated mesh sizes of more than 10 billion cells offer a difficult challenge for the data analysis applications. To provide adequate turnaround time, effort must be made to optimize each step in the analysis pipeline to squeeze out the optimal performance from the currently allocated resources.

Starting with a file format that can be efficiently processed in a parallelized HPC environment is the optimal first step. File formats such as NetCDF and XDMF are modern file formats that were specifically designed to provide peak input/output performance in an HPC environment. Often times, however, the simulation codes used to generate these datasets are more concerned with solving a particular engineering problem and less concerned about how the results are postprocessed. Unfortunately, many of the current simulation codes are using 20- or 30-year-old data formats that are in no way optimized for parallel processing. As a result, the scientific visualization applications are left to process these legacy file formats as efficiently as they can within the constraints of the file layout.

There are additional opportunities to maximize performance of the visualization applications. Tweaking the applications using various tricks and techniques to reduce the amount of time to process the data is an obvious first step. Finding those optimization tricks requires an intimate understanding of how the visualization applications work and recognizing where to look for opportunities to reduce processing time. There are also opportunities to reduce data processing time by understanding how the complex system of processes communicates within the allocated computational resources. Ensuring that the fastest communication path is chosen so that each processor within the application is exchanging information as fast as possible will also have a significant impact on processing time. Equally important is selecting the proper configuration of resources allocated to the data analysis process.

More of an art than exact science, understanding how much memory per processor and the total number of processors required to adequately process a large dataset can significantly impact the data analysis process. Allocating an insufficient amount of memory and/or processors will starve the application due to inadequate resources and the dataset will either fail to load or the total processing time will be greatly increased. Conversely, allocating too many resources can possibly increase processing time by creating a thrashing condition as the processors spend more time exchanging information about the status of the calculation rather than performing the actual calculation. Unfortunately, there is no general rule of thumb for predicting the amount of resources required to process these large datasets as each dataset has unique characteristics that impact processing time. Dr Anders Grimsrud from Computational Engineering, Inc (developer of the EnSight scientific visualization tool) has suggested a rule of thumb of 75–100 million cells per server as a starting point for estimating the amount of resources required to adequately visualize a dataset. Generally, some educated guesses can be made as a first attempt to evaluate processing time, and iterations of tests can help to narrow down the best processor and memory configuration.

A final opportunity for optimizing processing of Frontier datasets is at the image-rendering step. Typically, an HPC calculation is transient in nature, generally consisting of several hundred timesteps or more of data. Therefore, the amount of time spent generating an image can be significant as this rendering time is accumulated over hundreds or thousands of timesteps. Traditionally, using batch processing on the HPC systems has relied on software rendering to generate images. Using software rendering provides a substitution for hardware-based rendering (which would be available on a traditional end-user workstation) and for most moderately sized datasets software rendering performance is more than adequate. For the largest datasets such as those generated by HPC Frontier projects where

several hundred million polygons of data could be generated, additional functionality offered by distributed rendering alleviates many potential rendering bottlenecks that can be encountered by a single-processor graphics engine. Distributing the rendering process over multiple nodes allows each allocated processor to work on a portion of the geometry and the final image is composited to produce the final image.

For the purpose of this particular project, it was also of interest to determine how performance could be impacted by using graphics hardware resources. HPC compute nodes typically do not have graphics hardware that would normally be available on a traditional desktop workstation; however, the DOD HPCMP currently has a number of systems that offer compute nodes with graphics rendering hardware. Unfortunately, during the course of this particular project, there were numerous unexpected technical issues and administrative policies that prohibited the full use of enhanced graphics hardware rendering on the HPC systems. Enabling hardware rendering on HPC assets is an ongoing initiative and those results will be addressed in future projects.

2. Problem Description

Several datasets were made available for analysis during the course of this project. Those datasets were generated on DOD/HPC supercomputing assets in support of a particular Frontier project. This computational fluid dynamics simulation generated data in Plot3D format, a very common and well-tested structured data format developed by National Aeronautics and Space Administration (NASA)/Ames in 1990 (Pamela Walatka et al., NASA/Ames, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19900013774.pdf). Plot3D was not specifically developed for highly parallelized computing architectures; however, the visualization software efficiently handled the relatively simple structured data format. The sample datasets provided by the researcher team included the following:

- Set 1: 765 million cells, 8 blocks
- Set 2: 2.2 billion cells, 1 block
- Set 3: 2.2 billion cells, 64 blocks

These sample datasets were representative of the type of data that might be generated in the earliest stages of the Frontier-project development timeline and provided an opportunity to measure performance against known computational results. The Frontier-project research team also provided guidelines for what type of analysis would typically be performed on the computational results:

- Create isosurface of Q_Criterion function
- Create isosurface of shock wave boundary
- Generate 4K hi-resolution image (3,840 × 2,160 pixels)

Calculating the Q_Criterion value is a computationally expensive operation that needed to be derived from known values provided by the dataset. Moreover, the calculation and generation of the Q_Criterion isosuface proved to be a very difficult process. When processing the 765 million-cell dataset, the resulting isosurface of the Q_Criterion function consisted of over 150 million triangles. As it turns out, this geometry was larger than any single high-end graphics card could display and caused the visualization processes to crash. The 2.2 billion-cell dataset resulted in a Q_Criterion isosuface with more than 448 million triangles. Methods need to be developed and tested that would allow for geometry of this size to be properly rendered and saved out as an image. A sample of a resulting image showing isosurfaces of the Q_Criterion function and the shock wave boundary is shown in Fig. 1.

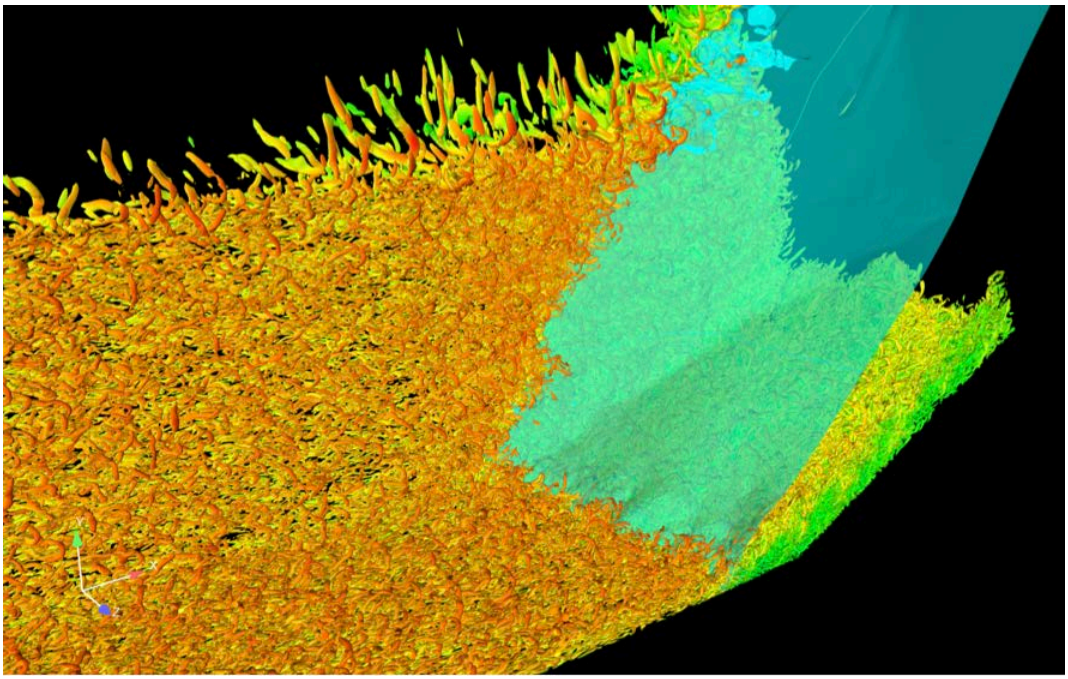

Fig. 1 Isosurface of Q_Criterion function and shock wave boundary generated using EnSight. Data courtesy of Dr Nicholas Bisek, US Air Force Research Laboratory/Aerospace Systems Directorate.

3. Methodology

At the time these datasets were made available, several common visualization tools were tested to see how they would handle datasets of this type and size. As mentioned previously, ParaView, VisIt and EnSight are the primary visualization software packages supported by the DOD/HPC DAAC. These packages were tested against the sample datasets starting in January 2015 and at that time EnSight was the closest to being able to complete the required tasks. In the earliest tests, an interactive EnSight session was able to load the entire dataset and calculate the Q_Criterion and shock wave boundary isosurfaces, but failed while rendering the final image. As mentioned previously, it was ultimately determined that the geometry required to render the scene was larger than could physically fit on a single workstation graphics card.

A relatively simple work-around for this geometry issue was to not load the entire structured dataset for interactive analysis. Because of the nature of the structured data format of this Plot3D data, EnSight was able to set a "stride" value (Fig. 2). Setting the stride value to 2 instructed the Plot3D data reader to process every other mesh value in the I, J, and K direction, reducing the problem size to one-eighth of the original size. Larger stride values reduce the amount of data processed incrementally, allowing for quick analysis and a rough estimate of the final, full-resolution result. It is not necessary to read in the dataset at full resolution to perform preliminary interactive analysis. All of the functionality from loading the data to calculating functions and generating isosurfaces are orders of magnitude faster when working with a decimated instance of the entire dataset. From this rough analysis of a subset of the data, it is then possible to save out EnSight commands scripts required to process the full dataset in batch mode.

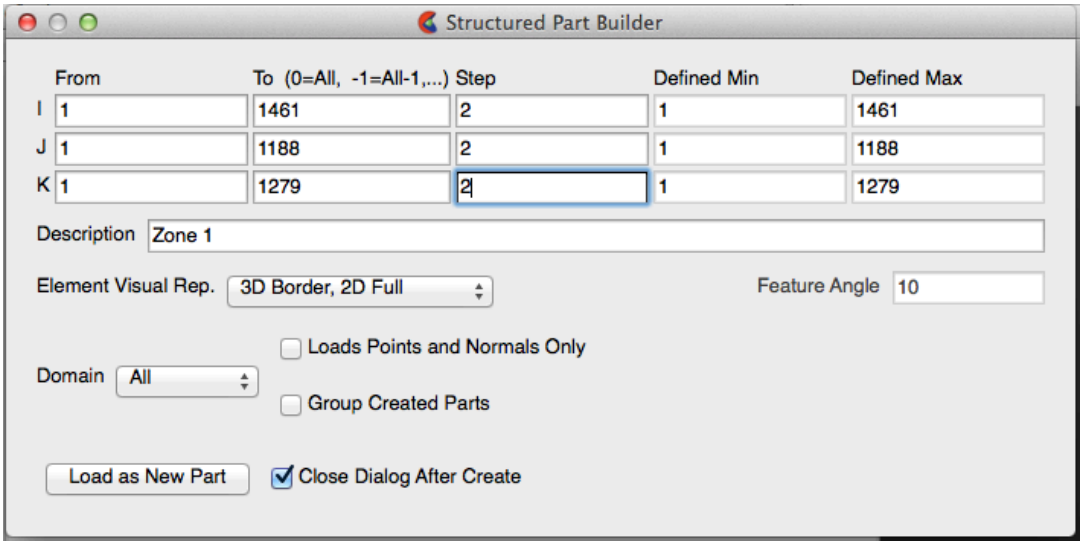

Fig. 2 The EnSight user interface provides an opportunity to set the step value for striding through a structured dataset

Once it was determined that EnSight could perform all of the required processing, a test plan was developed. Some of the critical objectives of this evaluation process included the following:

- Determine if EnSight can be scaled and optimized to visualize Frontier-sized HPC datasets on existing HPC resources
- Evaluate EnSight performance and find opportunities to optimize performance
- Evaluate alternative EnSight configurations
    - Impact of software-distributed rendering
    - Impact of hardware rendering if available
    - Locality of distributed rendering clients (DR Clients)
        - ✓ Performance of DR Clients running in the same nodes as the compute servers versus the DR Clients running on dedicated nodes

The intent of this evaluation was not only to determine the feasibility of using EnSight to support Frontier-project analysis but to optimize execution to achieve the best performance. Several opportunities to improve performance were discovered early on in the evaluation process. Instructing EnSight to load the computational mesh as “nonvisual” saved a significant amount of time. Traditionally, the computational mesh is not visualized; features such as clip planes or isosurfaces are extracted from the computational mesh and visualized. Allowing

EnSight to read the mesh geometry while treating it as nonvisual eliminates all the communication overhead of sending the geometry of the entire computational mesh from the allocated servers (where the data resides) to the client. Knowledge of this seemingly simple task saves an extraordinary amount of processing time and the savings is incremental when accumulated over each timestep of data that is processed.

Additional performance enhancements were obtained by instructing EnSight to use internal network interfaces to communicate between processes on the allocated compute nodes. EnSight is not an MPI-based parallel application and instead consists of a number of threaded applications and internode communication is handled via traditional transmission control protocol (TCP) socket communication. When establishing the infrastructure to support communication between nodes, the default hostname tends to be the public Ethernet connection that is shared by a variety of different system and user applications. After observing inconsistent performance results while re-running the same tests, it was discovered that the internode process communication was using the shared network interface rather than the private, high-speed interface. Changing the EnSight communication configuration to use the private network interface provided a substantial performance improvement and more consistent timing results.

4. EnSight Configuration

EnSight provides a significant amount of flexibility in determining how the various application client-side services and server-side computational processes are configured. This evaluation focused on 5 different configurations that were selected to assess a number of different technical implementations and demonstrate where performance improvements could be found. The number of nodes and processes per node was configured based on an estimate that would provide the best performance on the computational system where the initial testing was performed. The distribution of allocated resources is system dependent, based on the hardware configuration of the system compute nodes and the size of the computational mesh. For this particular dataset and hardware configuration, it was determined that 4 processors and 8 processes per node were appropriate for the 765 million-cell dataset.

The first configuration (Fig. 3) tested was also the most basic. In this example, the HPC job scheduling and resource allocation utility Portable Batch System (PBS) was tasked to allocate 4 compute nodes and 8 processes per node for each test. The EnSight client-side processes were started on the first allocated node. The EnSight server-side processes were started up on all 4 allocated nodes, with 8 servers

running on each node for a total of 32 data servers. In this example, the first allocated node had the burden of running all of the client-side processes along with the 8 server-side computational processes while the remaining 3 nodes ran only the 8 data servers. Depending on the size of the data and the amount of geometry that was generated, this may not have been an appropriate configuration due to memory and computational limitations, but was adequate for most moderate-sized datasets. The details of the underlying PBS batch script for this example are documented in Appendix A of this document.

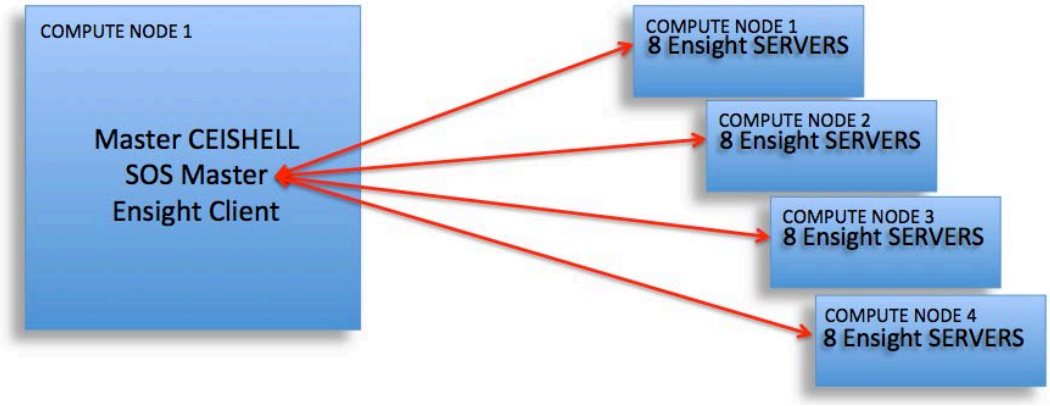

Fig. 3 Basic EnSight 4-node client-server configuration

The second configuration (Fig. 4) was very similar to the first configuration with the exception that PBS allocated a fifth node. This extra node was used to isolate the client-side and server-side processes to determine if there was any performance benefit to isolating the client-side render processes from the server-side compute-intensive processes. This configuration also alleviated any potential burdens on the first allocated node related to the process overload potential described in the first configuration. The details of the underlying PBS batch script for this example are documented in Appendix B of this document.

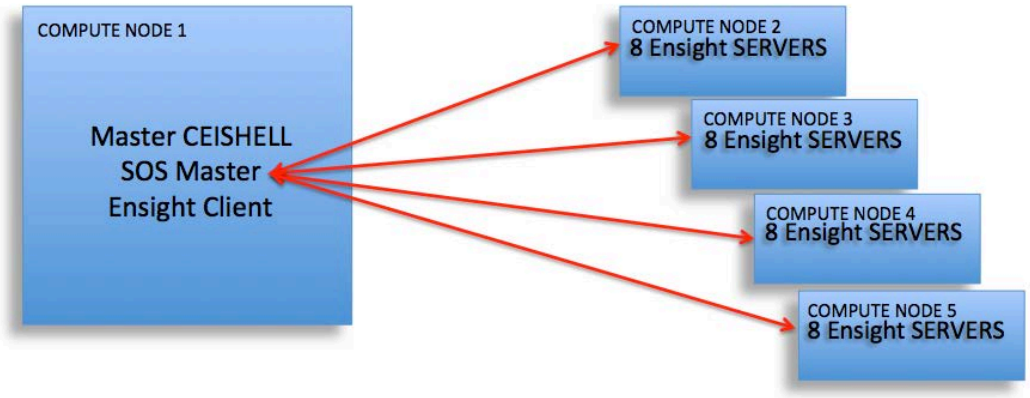

Fig. 4 EnSight 5-node client-server configuration

The third configuration (Fig. 5) tested was identical to the second configuration except that the client-side applications were running on a graphics node. This graphics node included nVidia graphics hardware and allowed EnSight to render the graphics objects using the dedicated nVidia graphics hardware rather than relying on software rendering as done in the first 2 configurations. One of the DOD/HPCMP utility server systems was modified to allow for mixed-node asset allocation (using both graphics and compute nodes in a single batch job). Using this mixed-node configuration, the client-side processes (including all of the graphics rendering) were dedicated to a single hardware-enabled graphics node. Hardware rendering should provide better performance than software rendering. The details of the underlying PBS batch script for this example are documented in Appendix C of this document.

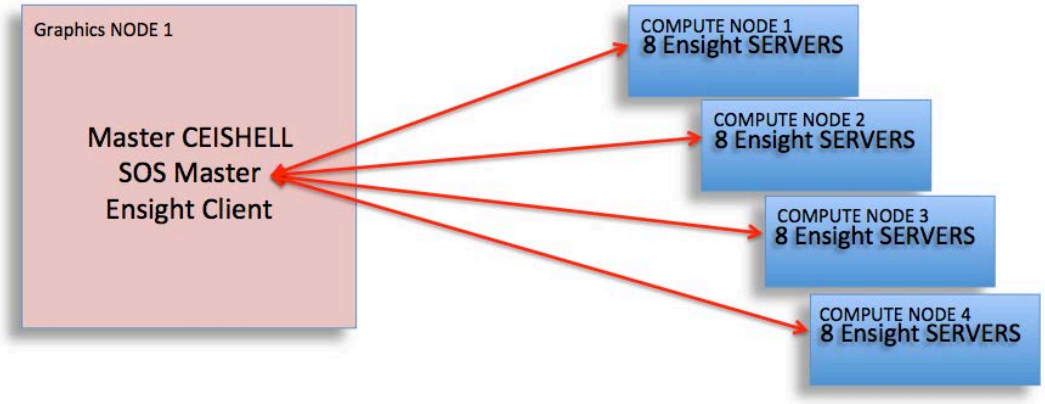

Fig. 5 EnSight 5-node client-server configuration using hardware-enabled client node

The fourth configuration was significantly more complicated because it implemented EnSight software-distributed rendering (also known as parallel compositing). As previously mentioned, distributed rendering allows multiple rendering clients to work on a subset of the geometry and the final image (consisting of partial images generated by each contributing distributed rendering client) is composited by a master collaborative rendering hub. The location of the distributed rendering client is highly configurable and in the example shown in Fig. 6, there is a single distributed rendering client running on each allocated node. However, there could be 4 software rendering clients running on a single node, 2 software rendering clients on each of the 4 allocated compute nodes for a total of 8 distributed renderers, or 3 rendering clients running on a unique node. As with other aspects of running EnSight in a parallel environment, there are numerous configuration options and many opportunities to tweak the configuration for a particular dataset on a particular computing resource. The details of the underlying PBS batch script for this example are documented in Appendix D of this document.

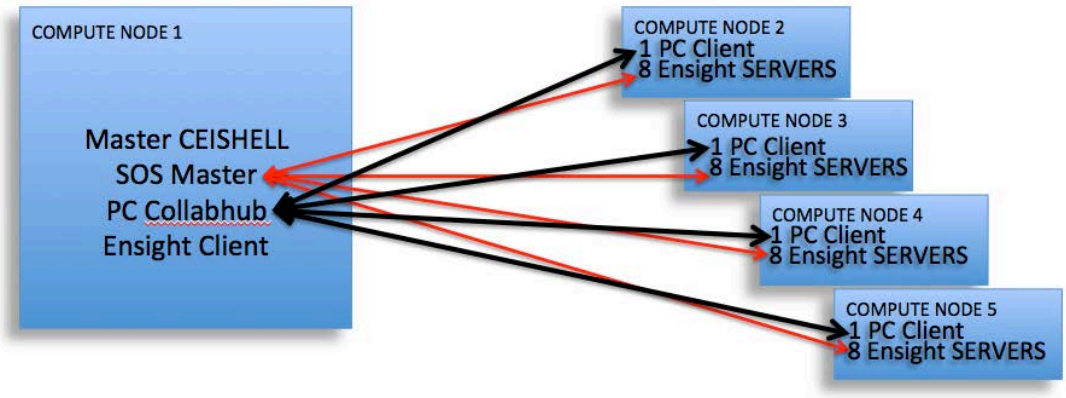

Fig. 6 EnSight distributed rendering configuration using standard compute nodes

The fifth configuration shown in Fig. 7 was the most complicated to set up and it had the potential to provide the best rendering performance, as all of the distributed rendering (along with final image compositing) would be performed using compute nodes with nVidia graphics acceleration. However, as stated earlier, numerous technical and administrative hurdles where encountered while trying to evaluate this configuration, and it was not possible to test this solution.

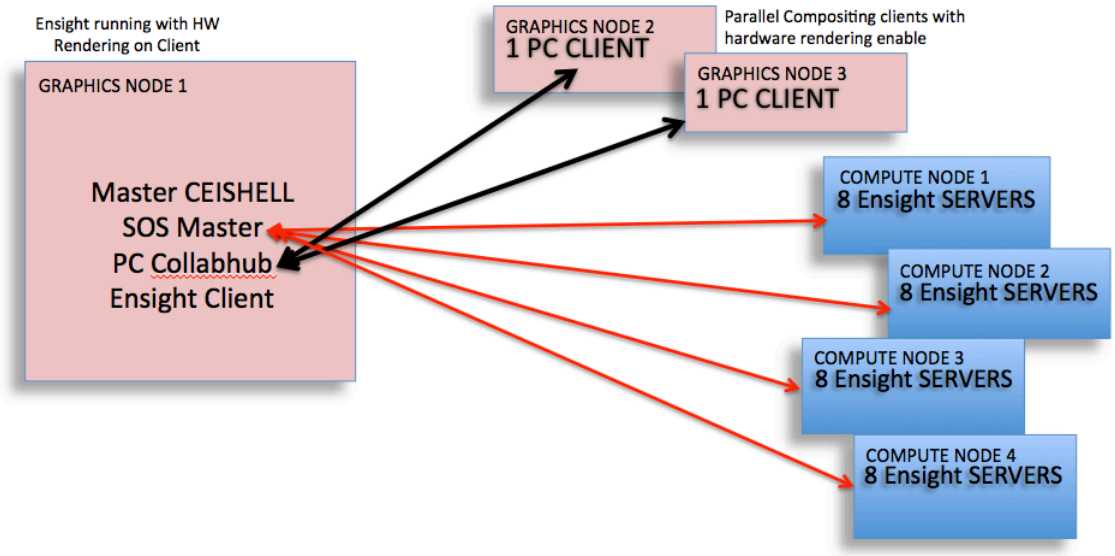

Fig. 7 EnSight distributed rendering configuration using hardware-enabled graphics nodes

For the sake of consistency, each configuration was tested using the same batch command script. The script read the same dataset each time, which remained on the same file system, and the same image was generated at the completion of each test configuration. EnSight was able to generate timing results for each phase of the visualization process and the output from EnSight related to the timing evaluation looked like the following:

Total elapsed time on client/seconds	TIME TO LOAD DATASET = 100.20
Total elapsed time on client/seconds	TIME TO ACTIVATE VARIABLES = 194.44
Total elapsed time on client/seconds	TIME TO CALCULATE ADDITIONAL VARIABLES = 1253.45
Total elapsed time on client/seconds	TIME TO CALCULATE ISOSURFACES = 46.66
Total elapsed time on client/seconds	TIME TO LOAD ISOSURFACES = 48.59
Total elapsed time on client/seconds	TIME TO SAVE IMAGE = 82.84

5. Results

Initial performance timing was done on the various installed versions of EnSight available on an HPC utility server using an otherwise unoptimized configuration and those exploratory results are shown in Fig. 8. In the case of EnSight versions 10.0.3c, 10.1.2a, and 10.1.2b, a distributed rendering option was not available and there were significant performance issues identified while generating the 4K images. EnSight 10.1.4a was under development at the time of this project, and the developers made code changes to this release as bugs and issues were identified. While overall performance improved with each incremental software version, there

was a substantial performance increase in the final release of EnSight 10.1.4a. The overall time to process the entire dataset was reduced from 1,985 to 672 s, while the 4K image-rendering step went from 1,050 to 260 s.

765M Cell Dataset Results	EnSight 10.0.3c	EnSight 10.1.2a	EnSight 10.1.2b	EnSight 10.1.4a
LOAD DATA	288 sec	260 sec	246 sec	137 sec
ACTIVATE VARIABLES	110	92	84	58
CALCULATE ADDITIONAL VARIABLES	377	279	270	141
CALCULATE ISOSURFACES	160	250	220	76
SAVE IMAGE 4K/4passes	1050	1000	860	260
TOTAL (SECONDS)	**1985**	**1881**	**1680**	**672**

Fig. 8 Timing results from various EnSight releases

In Fig. 9, the results from each of the 5 different configurations described earlier are shown. The value shown in this table is the amount of time (in seconds) to render and save a 4K image using the various configurations. In some of the configuration tests, there was no appreciable timing improvement. There is some noticeable improvement when the client-side processes are executed on a single graphics node. However, a much more dramatic improvement can be seen when using distributed software rendering. As mentioned earlier, the evaluation of parallel compositing using hardware-enabled distributed rendering on compute nodes with nVidia graphics cards could not be completed due to implementation and policy issues that have yet to be resolved.

Dataset	No Parallel Compositing (SW, 4 nodes)	No Parallel Compositing (SW, 5 Nodes)	No Parallel Compositing (HW, 5 Nodes)	Parallel Compositing (SW Rendering)	Parallel Compositing (HW Rendering)
765M	607 sec	605 sec	547 sec	260 sec	TBD
2.2B/1 block	1711 sec	1671sec	1604 sec	308 sec	TBD
2.2B/64block	1696 sec	1707 sec	1617 sec	567sec	TBD

Fig. 9 Time to render a 4K image using various configurations against 3 sample datasets

Finally, the performance tests were run on a number of different HPC compute systems. While the bulk of this project's effort was focused on the utility servers, it was possible to do testing on other platforms using distributed software rendering. Figure 10 shows the results of running similar tests across different platforms.

SAMPLE DATA SET	765M Cells/8 Blocks	2.2B Cells/1Block	2.2B Cells/64 Blocks
UTILITY SERVER (Appro Xtreme-X)			
WITHOUT Distributed Rendering	607 sec	1696 sec	1711 sec
Software Distributed Rendering	260 sec	308 sec	567 sec
PERSHING (IBM iDataPlex)			
WITHOUT Distributed Rendering	352 sec	*FAILED*	*FAILED*
Software Distributed Rendering	163 sec	188 sec	263 sec
GARNET (CRAY XE6)			
WITHOUT Distributed Rendering	590 sec	*FAILED*	*FAILED*
Software Distributed Rendering	100 sec	99 sec	162 sec
EXCALIBUR (CRAY XC40)			
WITHOUT Distributed Rendering	291 sec	774 sec	777 sec
Software Distributed Rendering	106 sec	94 sec	163 sec

Fig. 10 Timing results from a variety of DOD/HPCMP supercomputer resources

The performance value of software-distributed rendering is apparent on these other platforms but this also demonstrates how distributed rendering can be essential for Frontier-sized datasets. Those blocks on Pershing and Garnet that are marked as "FAILED" indicate that the resulting geometry could not fit on a single compute node and therefore could not possibly be rendered without using distributed rendering/parallel compositing. The best result for distributed software rendering was on the new DOD/HPCMP supercomputer system called Excalibur, a Cray XC40 computer system located at the US Army Research Laboratory at Aberdeen Proving Ground, MD. The Excalibur system does have graphics-enable compute nodes and future testing will include evaluation of distributed hardware rendering.

6. Conclusion

EnSight is a commercially available scientific visualization package provided by Computational Engineering, Inc. This software application has been used extensively at the US Army Research Laboratory and across the DOD/HPCMP for more than 20 years. EnSight has been established as an essential application for interactive data analysis of large HPC simulation results. This particular project focused on not only being able to produce results from very large Frontier-class computational data results but to also optimize overall performance to reduce the amount of time required to visualize those results.

When working on Frontier-class datasets, consideration needs to be given to the amount of resulting geometry generated by the visualization process. When processing multibillion cell computation meshes, it is possible, and even likely, to generate more geometry than can physically fit on a single graphics card or in memory on a single computational node. In these instances, distributed rendering (or parallel compositing) is essential functionality required to support Frontier-class datasets. Using hardware rendering on modestly sized datasets can significantly improve performance and there is future work to be done to evaluate distributed hardware rendering as applied to these multibillion cell datasets. However, as demonstrated in this project, the implementation of distributed software rendering provided essential and significant performance enhancements necessary to complete the stated requirements of the Frontier project computational researchers who contributed the sample datasets.

Finally, EnSight provides a significant amount of flexibility in the distribution of functional processes across the allocated high-performance computing resources. The configuration of client-side rendering and data services, server-side computational servers, and the allocation of distributed rendering servers allows the application analyst to consider many different configurations, mitigating potential oversubscription of resources and providing the most appropriate configuration to allow for the visualization of a particular dataset.

7. References

Angelini R. EnSight HPC job launching. DOD HPC InSights. Spring 2011. p. 10.

Angelini R. Client-Server HPC job launching. DOD HPC InSights. Spring 2012. p. 3.

Walatka P, Buning P, Pierce L, Elson P. PLOT3D User's Manual. c2015 May 27 [accessed 2015 July 06]. http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19900013774.pdf.

INTENTIONALLY LEFT BLANK.

Appendix A. Simple 4-node Batch Example Script

This is a very basic EnSight batch command file used on a High-Performance Computing Program Modernization Program Utility Server system. This Portable Batch System (PBS) batch script allocates 4 compute nodes with 8 processors per node for a total of 32 allocated processes. The EnSight client will run on the first allocated node, and the 32 EnSight servers will be distributed across all 4 allocated nodes.

```
#!/bin/csh
```

Required PBS resource flags

```
#PBS -N simple_batch
#PBS -A Project_ID_Removed
#PBS -q serial
#PBS -l select=4:ncpus=16:mpiprocs=8:mem=250GB
#PBS -l walltime=2:00:00
#PBS -j oe
#PBS –l application=ensight
```

Create a job-specific subdirectory based on JOBID and move into it

```
set JOBID=`echo ${PBS_JOBID} | cut -d '.' -f 1`
set wkdir=${WORKDIR}/${JOBID}
mkdir -p ${wkdir}
cd ${wkdir}
```

Load EnSight module to establish environment and copy user-provided scripts into working directory. Environment variable to increase internal buffer size to support 4K image render.

```
module load ensight/10.1.4b
setenv CEI_4K_BUFFER 1


cp ~/scripts/ensight.ctx* .
cp ~/scripts/num_elements_nodes.py .
cp ~/scripts/ensight_script.enc .
```

Find out how many processor were allocated to this batch job

```
cat $PBS_NODEFILE
set NP=`wc -l $PBS_NODEFILE | cut -f1 -d" "`
echo NUMBER OF PROCS = $NP
```

The node where the program starts executing is the master node and needs to be passed to the server startup command. Make sure to use the private Infiband network interface rather than the default Ethernet interface. The Infiband interface is designated in the form of hostname-n3.

```
set master=`hostname -s`-n3
```

Create a random number to be used to establish unique port numbers for client/server communication

```
set random=0
while ($random < 10000)
      set x="`date +%N` % 65000"
      set random=`echo $x | bc`
           echo $random
end

set CLIENT_PORT=$random
@ SERVER_PORT= ${CLIENT_PORT} + 1
echo $CLIENT_PORT $SERVER_PORT
```

Start up the EnSight client on the first allocated … listening for a ceishell connection on ${CLIENT_PORT}

```
${CEI_HOME}/bin/ceishell30 -app -v -end_after_ensight –child \
listen://\?port=${CLIENT_PORT}\&timeout=-1 &
```

Start up Server-of-Servers (SoS) on First allocated Node. Listening for communication from the EnSight servers on ${SERVER_PORT} and passing traffic back to the EnSight client via ${CLIENT_PORT}

```
echo "Starting up SoS Process on first allocated node"

${CEI_HOME}/bin/ceishell30 -parent
connect://${master}\?port=${CLIENT_PORT}\&timeout=-1 \
-child listen://\?nconnections=${NP}\&timeout=-1\&port=${SERVER_PORT} -role SOS &
```

Start up EnSight servers on each of the allocated nodes. Route all of the communication back to the SoS process via ${SERVER_PORT}

```
echo "Start up servers on all allocated nodes using mpirun"

mpirun -np ${NP} ${CEI_HOME}/apex30/machines/linux_2.6_64/ceishell \
-parent connect://${master}\?timeout=-1\&port=${SERVER_PORT} -role SOS_SERVERS  &
```

Start up EnSight client on the first allocated node and connect to ceishell communication network

```
ensight100 -X -batch -sos -ceishell -p ${cmd_file}
```

INTENTIONALLY LEFT BLANK.

Appendix B. Simple 5-node Batch Example Script

This Portable Batch System (PBS) batch command file is similar to the example from Appendix A except in this example the script allocates 5 compute nodes with 8 processors per node for a total of 32 allocated processes. The EnSight client will run on the first allocated node, and the 32 EnSight servers will be equally distributed across the remaining 4 nodes. There is additional work required in this startup script to identify the first allocated node and remove that node from the pool of resources available to run the server-side processes.

```
#!/bin/csh
```

Required PBS resources flags. In this instance, PBS is instructed to request 5 nodes in the "select" directive

```
#PBS -N 5Node_Batch
#PBS -A Project_ID_Removed
#PBS -q serial
#PBS -l select=5:ncpus=16:mpiprocs=8:mem=250GB
#PBS -l walltime=2:00:00
#PBS -j oe
#PBS -l application=ensight
```

Create a job-specific subdirectory based on JOBID and move to that directory

```
set JOBID=`echo ${PBS_JOBID} | cut -d '.' -f 1`
set wkdir=${WORKDIR}/${JOBID}
mkdir -p ${wkdir}
cd ${wkdir}
```

Load the EnSight module to establish environment and copy user-provided scripts into working directory

```
module load ensight/10.1.4b
setenv CEI_4K_BUFFER 1

set cmd_file="1.enc"
cp ~/Ens_DR/${cmd_file} .
cp ~/Ens_DR/num_elements_nodes.py .
```

Need to generate a new host resources file to be used by MPIRUN later in the script. Remove the "current" node from the available host list.

```
cat $PBS_NODEFILE > pbs.nodefile

grep -v `hostname -s` $PBS_NODEFILE > pbs.nodefile2
set NP=`cat pbs.nodefile2 | wc -l | cut -f1 -d" "`
echo NUMBER OF PROCS = $NP sort -u pbs nodefile2 > unique_nodefile
```

FORCE the Infiband Interface

```
set master=`hostname -s`-n3
```

Generate a random number to be used to establish unique port numbers for client/server communication

```
set random=0
while ($random < 10000)
      set x="`date +%N` % 65000"
      set random=`echo $x | bc`
           echo $random
end
set CLIENT_PORT=$random
@ SERVER_PORT= ${CLIENT_PORT} + 1
echo "PORTS=" $CLIENT_PORT $SERVER_PORT
```

Start up the MASTER EnSight client on the first allocated node. Listen for inter-process communication on ${CLIENT_PORT}

```
echo "Starting MASTER ceishell shell process on the first allocated node"
${CEI_HOME}/bin/ceishell31 -X -app -end_after_ensight -child
listen://${master}\?port=${CLIENT_PORT}\&timeout=-1 &
```

Start up the EnSight Server-of-Servers (SOS) process on the first allocated node. Listen for communication from the servers on ${SERVER_PORT} and pass results back to the EnSight master client on ${CLIENT_PORT}.

```
echo "Starting up SoS/CollabHUB process on first allocated node"
${CEI_HOME}/bin/ceishell31 -parent
connect://${master}\?port=${CLIENT_PORT}\&timeout=-1 \
-child listen://\?nconnections=${NP}\&timeout=-1\&port=${SERVER_PORT} -role SOS &
```

Start up EnSight servers on the remaining 4 nodes. Note the use of the "-hostfile pbs.nodefile2" flag to mpirun to use a subset of the originally assigned nodes. Pass all traffic back to the master SoS process via ${SERVER_PORT}.

```
echo "Start up servers on all allocated nodes using mpirun"
mpirun -hostfile pbs.nodefile2  ${CEI_HOME}/apex31/machines/linux_2.6_64/ceishell \
-parent connect://${master}\?timeout=-1\&port=${SERVER_PORT} -role SOS_SERVERS  &
```

Check the ceishell communication network. This command is very useful for debugging purposes.

```
echo "Display the ceishell network"
${CEI_HOME}/bin/ceishell31 -cmd show_net
```

Start up the EnSight client and pass the user-provided batch command script that will be automatically executed. EnSight will automatically connect to the ceishell communication infrastructure and start up the appropriate underlying application executable programs.

```
ensight101 -X -batch -sos -ceishell -p ${cmd_file}
```

INTENTIONALLY LEFT BLANK.

Appendix C. Mixed-Node Batch Script Example Script

This Portable Batch System (PBS) batch command file is similar to the 5-node example from Appendix B except in this example the directives request a mixture of graphics processing unit (GPU)-enabled and large memory compute nodes. The EnSight client will run on the first allocated node (which is the graphics node), and the 32 EnSight servers will be equally distributed across the remaining 4 nodes. There is additional work required in this startup script to identify the first allocated node and remove that node from the pool of resources available to run the server-side processes. There is also a section of code required to properly initialize the X-windows environment on the allocated graphics node.

```
#!/bin/csh
```

Required PBS resources flags. The "select" directive is requesting mixed nodes assigned to a single batch job—1 GPU-enabled node and 4 large memory compute nodes. The "develop" queue was a limited-access resource set up specifically to test hardware rendering on the GPU-nodes.

```
#PBS -N Mixed_Node
#PBS -q develop
#PBS -A Project_ID_Removed
#PBS -l select=1:ngpus=1 ncpus=16:mpiprocs=1+4 ngpus=0:mpiprocs=8 mem=250GB
#PBS -l walltime=02:00:00
#PBS -j oe
#PBS -l application=ensight
```

Create a job-specific subdirectory based on JOBID and move to that directory

```
set JOBID=`echo ${PBS_JOBID} | cut -d '.' -f 1`
set wkdir=${WORKDIR}/${JOBID}
mkdir -p ${wkdir}
cd ${wkdir}
```

Load the EnSight module to establish environment and copy user-provided scripts into working directory

```
module load ensight/10.1.4b
setenv CEI_4K_BUFFER 1

set cmd_file="1.enc"
cp ~/Ens_DR/${cmd_file} .
cp ~/Ens_DR/num_elements_nodes.py .
```

Need to generate a new host resources file to be used by MPIRUN later in the script. Remove the "current" node from the available host list.

```
echo PBS_NODEFILE
cat $PBS_NODEFILE
grep -v `hostname -s` $PBS_NODEFILE > pbs.hostname2
echo pbs.hostname2
cat pbs.hostname2
set NP=`cat pbs.hostname2 | wc -l | cut -f1 -d" "`
echo NUMBER OF PROCS = $NP
```

Set internode communication to use the Infiband private network Interface

```
set master=`hostname -s`-n3
```

Generate a random number to be used to establish unique port numbers for client/server communication

```
set random=0
while ($random < 10000)
      set x="`date +%N` % 40000"
      set random=`echo $x | bc`
           echo $random
end

set CLIENT_PORT=$random
@ SERVER_PORT= ${CLIENT_PORT} + 1

echo $CLIENT_PORT $SERVER_PORT
```

These commands were necessary to set up the proper environment on the GPU-enabled graphics node. This code is similar to what is done in some of the underlying SRD startup scripts and is very specific to the HPC utility servers.

```
echo "HOSTNAME="`hostname -s`
set XCOOKIESRD=/tmp/vgl_xauth_key

set XAUTHORITY=${HOME}/.Xauthority

#  Pull the display from the SRD x11 cookie.
set DISP=`xauth -q -f $XCOOKIESRD list | cut -d " " -f 1`
setenv DISPLAY :0
echo "DISPLAY=$DISPLAY"

ls -l $XCOOKIESRD
getfacl $XCOOKIESRD

# Merge in SRD generated cookie.
xauth -f $XAUTHORITY merge $XCOOKIESRD

# Verify direct rendering is enabled
echo "before glxinfo"

glxinfo > /dev/null
if ($status != 0) then
```

```
        echo "NO VALID DISPLAY ... EXITING NOW"
        exit 1
endif
glxinfo | grep -i "direct rendering"
echo "after glxinfo"
```

Start up EnSight client on the first allocated node and connect to ceishell communication network. Listen for inter-process communication on ${CLIENT_PORT}

```
${CEI_HOME}/bin/ceishell31 -app -v -end_after_ensight -child
listen://\?port=${CLIENT_PORT}\&timeout=-1 &
echo "DISPLAY=$DISPLAY"
```

Start up the EnSight Server-of-Servers (SOS) process on the first allocated node. Listen for communication from the servers on ${SERVER_PORT} and pass results back to the EnSight master client on ${CLIENT_PORT}.

```
echo "Starting up SoS Process on first allocated node"
${CEI_HOME}/bin/ceishell31 -parent connect://${master}\?port=${CLIENT_PORT}
\&timeout=-1 \
-child listen://\?nconnections=${NP}\&timeout=-1\&port=${SERVER_PORT} -role SOS &
echo "DISPLAY=$DISPLAY"
```

Start up EnSight servers on the remaining 4 nodes. Note the use of the "-hostfile pbs.nodefile2" flag to mpirun to use a subset of the originally assigned nodes. Pass all traffic back to the master SoS process via ${SERVER_PORT}.

```
echo "Start up servers on all allocated nodes using mpirun"
echo "DISPLAY=$DISPLAY"

mpirun -hostfile pbs.hostname2  ${CEI_HOME}/apex31/machines/linux_2.6_64/ceishell \
-parent connect://${master}\?timeout=-1\&port=${SERVER_PORT} -role SOS_SERVERS  &
echo "DISPLAY=$DISPLAY"
```

Start up the EnSight client and pass the user-provided batch command script that will be automatically executed. EnSight will automatically connect to the ceishell communication infrastructure and start up the appropriate underlying application executable programs. The command line argument "-X" has been removed from this startup command; removing this flags forces EnSight to do hardware-enabled rendering.

```
ensight101 -batch -sos -ceishell -p ${cmd_file}
# remove "-X" flag for hardware rendering!
```

After EnSight has completed the batch job, clean up the X-windows environment on the graphics node.

```
# Remove the cookie on shutdown.
xauth -f $XAUTHORITY remove $DISP
```

Appendix D. Cray Distributed Rendering Example Script

This is a complex sample Portable Batch System (PBS) batch script used to launch EnSight with distributed rendering in batch mode on a High-Performance Computing Modernization Program Cray supercomputer. In this example, PBS is asked to allocate 9 compute nodes. The first allocated node is reserved for the client processes, the Server of Servers (SOS) master process, and the distributed rendering (DR) collaborative hub. The EnSight compute-side servers are allocated to start on the remaining 8 nodes allocated by PBS. In this example, 4 distributed rendering clients will be used—a single DR process on each of the first 4 allocated server nodes.

```
#!/bin/csh
```

Required PBS resource flags

```
#PBS -N 8_nodes_8dr
#PBS -A Project_ID_Removed
#PBS -q standard
#PBS -l walltime=03:00:00
#PBS -j oe
#PBS -l select=9:ncpus=32:mpiprocs=8
#PBS -l ccm=y
```

How many server nodes will also run a single DR node?

```
set DR_NODES=4
```

Move to the system working directory

```
set WORKDIR=/p/work1/angel
cd ${WORKDIR}
```

Create a job-specific subdirectory based on JOBID and move to that directory

```
set JOBID=`echo ${PBS_JOBID} | cut -d '.' -f 1`
if ( ! -d ${JOBID} ) mkdir -p ${JOBID}
cd ${JOBID}
```

Establish EnSight execution environment

```
setenv CEI_HOME /usr/cta/DAAC/CEI/10.1.4b
set path=($CEI_HOME/bin $path)
setenv TMPDIR $WORKDIR/$JOBID
```

Identify required input files

```
set cmd_file="1.enc"
cp ~/DR/${cmd_file} .
cp ~/DR/num_elements_nodes.py .
```

Save off complete PBS node list.

```
cat $PBS_NODEFILE > pbs.nodefile
```

Determine how many processors per node were allocated

```
@ procs_node = `wc -l pbs.nodefile | cut -f1 -d" "` / `sort -u pbs.nodefile | wc -l `
echo procs_node = $procs_node
```

Remove first node from resource list-first node reserved for client-side processes

```
set head_node=`head -1 $PBS_NODEFILE`
grep -v $head_node $PBS_NODEFILE > server nodes
```

Save off file with the remaining nodes allocated for servers/DR

```
set TOTAL_SERVERS=`cat server.nodes | wc -l | cut -f1 -d" "`
echo TOTAL SERVERS = $TOTAL_SERVERS
```

Find out how many unique nodes we have to use for distributed rendering

```
sort -u server nodes > unique.nodefile
set UNIQUE_SERVERS=`cat unique.nodefile | wc -l | cut -f1 -d" "`
echo UNIQUE_SERVERS = $UNIQUE_SERVERS
```

Find out if we asked for more unique DR nodes than we have available

```
if ($UNIQUE_SERVERS < $DR_NODES) then
        echo SOMETHING IS WRONG - not enough nodes allocated
        exit 0
 endif
echo NUMBER OF DR PROCS = $DR_NODES
```

Create a random number that EnSight will use to generate unique port numbers

```
set random=0
while ($random < 10000)
      set x="`date +%N` % 40000"
      set random=`echo $x | bc`
end

set SERVER_PORT=$random
@ DR_PORT = $random + 1
```

Make sure client-side batch script does not exist

```
set sf="serial_startup.csh"
if (-e ${sf}) rm ${sf}
```

Create batch command script to start up serial processes on first allocated node

```
cat << EOF > ${sf}
#!/bin/csh

setenv TMPDIR $WORKDIR
setenv CEI_HOME ${CEI_HOME}
set path=(\${CEI_HOME}/bin \$path)

setenv ENSIGHT10_MAX_THREADS 4
setenv ENSIGHT10_MAX_SOSTHREADS 8
setenv ENSIGHT10_MAX_CTHREADS 8
setenv ENSIGHT10_SOCKBUF 1048576
setenv CEI_4K_BUFFER 1

cd ${WORKDIR}/${JOBID}

if (-e master node) rm master node
hostname -s > master.node
echo Master node processes running on \`hostname -s\`

# start up EnSight and have it look for a CEISHELL network to connect to

( ${CEI_HOME}/bin/ensight101 -X -prdist -batch -sos -ceishell -p ${cmd_file} )&

echo ENSIGHT CLIENT STARTED

# Start up CEISHELL network for client-side services

( ${CEI_HOME}/bin/ceishell31 -X -app -end_after_ensight \
-child listen://\?nconnections=${TOTAL_SERVERS}\&timeout=-1\&port=${SERVER_PORT} \
-role SOS \
-child listen://\?nconnections=${DR_NODES}\&timeout=-1\&port=${DR_PORT} \
-role COLLABHUB ) &

echo MASTER/SOS/COLLABHUB CEISHELL started

# Query the CEISHELL network and to show all of the interprocess connections
(${CEI_HOME}/bin/ceishell31 -cmd show_net) &

wait
EOF
```

Set permissions on the client-side startup script and display the contents of the file

```
chmod 755 ${sf}
echo "\n============================\n"
cat ${sf}
echo "\n============================\n"
```

Make sure the server-side script file does not exist

```
set df="dr_startup.csh"
if (-e ${df}) rm ${df}
```

Create batch command script to start up SoS/DR processes on remaining allocated nodes

```
cat << EOF > ${df}
#!/bin/csh

setenv TMPDIR $WORKDIR
setenv CEI_HOME ${CEI_HOME}
set path=(\${CEI_HOME}/bin \$path)

setenv ENSIGHT10_MAX_THREADS 4
setenv ENSIGHT10_MAX_SOSTHREADS 8
setenv ENSIGHT10_MAX_CTHREADS 8
setenv ENSIGHT10_SOCKBUF 1048576
setenv CEI_4K_BUFFER 1

cd ${WORKDIR}/${JOBID}

set master_node=\`cat master.node\`
echo Starting up SoS/DR processes on \$master_node

echo STARTING APRUN  SOS_SERVERS

#pick up command line argument
set DO_DR=\$1

# main job launching loop.   Start up an SoS server on each node & processor allocated.
# Start up a single DR client on each unique node if needed
foreach i (\`seq 1 ${procs_node}\` )
echo inside loop \$i on `hostname -s`
   echo start up SOS_SERVER on \`hostname -s\`
   ( ${CEI_HOME}/apex31/machines/linux_2.6_64/ceishell \
    -parent connect://\${master_node}\?port=${SERVER_PORT}\&timeout=-1 -role
SOS_SERVERS ) &
   if (\$i == 1 && \$DO_DR == 1) then
           echo STARTED DR_CLIENT on \`hostname -s\`
           ( ${CEI_HOME}/apex31/machines/linux_2.6_64/ceishell \
           -parent connect://\${master_node}\?port=${DR_PORT}\&timeout=-1 -role
DRCLIENTS -X) &
   endif
end

# Don't terminate this process until all of the background processes have completed
wait
EOF
```

Set permissions on the server-side startup script and display the contents of the file

```
chmod 755 ${df}
echo "\n============================\n"
cat ${df}
echo "\n============================\n"
```

Use APRUN to submit batch command script to the first compute node

```
( aprun -n 1 -d 1  ./${sf} ) &
sleep 5
```

Fire up the EnSight server executable on the allocated compute nodes using APRUN. Apparently, APRUN is smart enough to not re-use the first node and will use the remainder of the nodes only. Each iteration of this loop starts on a new node.

```
set dr_counter=1
foreach i (`seq 1 $UNIQUE_SERVERS`)
  echo loop $i
  if ($dr_counter <= $DR_NODES) then
        echo starting up aprun with DR
             (aprun -n 1  -N 1 ./${df} 1 ) &
  else
        echo starting up aprun without DR
             (aprun -n 1  -N 1 ./${df} 0 ) &
  endif
  @ dr_counter = $dr_counter + 1
end
```

Don't terminate this shell script until all of the background processes have completed

```
wait
echo " APRUN DONE `date`"
```

Bibliography

DOD HPC Frontier Projects. DOD HPC Modernization Program; c2014 [accessed 2015 July 6] http://www.hpc.mil/index.php/2013-08-29-16-04-43/resource-management/frontier-projects.

DAAC Data Analysis and Assessment Center. DOD HPC Modernization Program; [accessed 2015 July 6] HPCMP DAAC Website: http://daac.hpc.mil.

VisIt Website. Livermore (CA): Lawrence Livermore National Laboratory; [accessed 2015 July 6] https://wci.llnl.gov/simulation/computer-codes/visit.

Welcome to ParaView. ParaView; [2015 July 6] http://www.paraview.org.

EnSight 10.1. EnSight CSM and CFD Post processing; c2014 [accessed 2015 July 6] http:// www.ceisoftware.com.

Main Page. XDMF; 2014 Nov 7 [2015 July 6] http://www.xdmf.org.

Network Common Data Form (NetCDF). Boulder (CO): Unidata; c2015 [accessed 2015 July 6] http://www.unidata.ucar.edu/software/netcdf.

INTENTIONALLY LEFT BLANK.

List of Symbols, Abbreviations, and Acronyms

DAAC	Data Analysis and Assessment Center
DOD	Department of Defense
DR Clients	distributed rendering clients
GPU	graphics processing unit
HPC	high-performance computing
HPCMDC	High-Performance Computing Modernization Program
NASA	National Aeronautics and Space Administration
PBS	Portable Batch System
TCP	transmission control protocol

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

3 (PDF) DIR USARL RDRL CIH S R ANGELINI D SHIRES L BRAINARD